An Out-of-the-box Full-network Embedding for Convolutional Neural Networks

Dario Garcia-Gasulla*, Armand Vilalta*, Ferran Parés*
Eduard Ayguadé*†, Jesus Labarta*†, Ulises Cortés*†, Toyotaro Suzumura*‡
*Barcelona Supercomputing Center (BSC)
Barcelona, Spain
Email: dario.garcia@bsc.es, armand.vilalta@bsc.es
†Universitat Politècnica de Catalunya - BarcelonaTECH
Barcelona, Spain
‡IBM T.J. Watson
New York, USA

***Abstract*—Features extracted through transfer learning can be used to exploit deep learning representations in contexts where there are very few training samples, where there are limited computational resources, or when the tuning of hyper-parameters needed for training deep neural networks is unfeasible. In this paper we propose a novel feature extraction embedding called full-network embedding. This embedding is based on two main points. First, the use of all layers in the network, integrating activations from different levels of information and from different types of layers (*i.e.*, convolutional and fully connected). Second, the contextualisation and leverage of information based on a novel three-valued discretisation method. The former provides extra information useful to extend the characterisation of data, while the later reduces noise and regularises the embedding space. Significantly, this also reduces the computational cost of processing the resultant representations. The proposed method is shown to outperform single layer embeddings on several image classification tasks, while also being more robust to the choice of the pre-trained model used as transfer source.**

***Index Terms*—Transfer Learning, Feature Extraction, Embedding Spaces**

I. Introduction

Deep learning models, and particularly convolutional neural networks (CNN), have become the standard approach for tackling image processing tasks. The key to the success of these methods lies in the rich representations deep models build, which are generated after an exhaustive and computationally expensive learning process [1]. To generate deep representations, deep learning models have strong training requirements in terms of dataset size, computational power and optimal hyper-parametrisation. For any domain or application in which either of those factors is an issue, training a deep model from scratch becomes unfeasible.

Dario Garcia-Gasulla, Armand Vilalta and Ferran Parés contributed equally to this work. This work is partially supported by the Joint Study Agreement no. W156463 under the IBM/BSC Deep Learning Center agreement, by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project and by the Generalitat de Catalunya (contracts 2014-SGR-1051), and by the Core Research for Evolutional Science and Technology (CREST) program of Japan Science and Technology Agency (JST).

Within deep learning, the field of transfer learning studies how to extract and reuse pre-trained deep representations. This approach has three main applications: improving the performance of a network by initialising its training from a non-random state [2]–[4], enabling the training of deep networks for tasks of limited dataset size [5], [6], and exploiting deep representations through alternative machine learning methods [7]–[9]. The first two cases, where training a deep network remains the end purpose of the transfer learning process, are commonly known as *transfer learning for fine-tuning*, while the third case, where the end purpose of the transfer learning does not necessarily include training a deep net, is typically referred as *transfer learning for feature extraction*.

Of the three limiting factors of training deep networks (*i.e.*, dataset size, computational cost, and optimal hyper-parametrisation), transfer learning for fine-tuning partly solves the first. Indeed, one can successfully train a CNN on a dataset composed by roughly a few thousand instances using a pre-trained model as starting point, and achieve state-of-the-art-results. Unfortunately, fine-tuning a model still requires a dataset size in the order of thousands, a significant amount of computational resources, and lots of time and insight to optimise the multiple hyper-parameters involved in the process.

Transfer learning for feature extraction on the other hand is based on processing a set of data instances through a pre-trained neural network by doing a feed-forward pass, extracting the neural activation values to generate a new data representation which can then be used by another learning mechanism. This is applicable to datasets of any size, as each data instance is processed independently. It has a relatively small computational cost, since there is no deep net training. And finally, it requires no hyper-parameter optimisation or tuning of any sort. Significantly, the applications of transfer learning for feature extraction are limited only by the capabilities of the methods that one can execute on top of the generated deep representations, and addresses the needs of many industrial applications where labelled data availability is often scarce.

II. Related Work

Transfer learning studies how to extract and reuse deep representations learnt for a given task `t0`, to solve a different task `t1`. Fine-tuning approaches require the `t1` target dataset to be composed by at least a few thousands instances, to avoid overfitting during the fine-tuning process. To mitigate this limitation, it has been proposed to reuse carefully selected parts of the `t0` dataset in the fine-tuning process alongside the `t1` (*i.e.*, selective joint fine-tuning) [5], and also to use large amounts of noisy web imagery alongside with clean curated data [10]. In fine-tuning, choosing which layers of weights from the `t0` model should be transferred, and which should be transferred and kept unchanged on the `t1` training phase has a large impact on performance. Extensive research on that regard has shown that the optimal policy depends mostly on the properties of both `t0` and `t1` [7], [11], [12]. This dependency, together with the hyper-parameters inherent to deep network training, defines a large search space to be explored by fine-tuning solutions.

Given a pre-trained model for `t0`, instead of fine-tuning a deep model, one may use alternative machine learning methods for solving `t1`. For that purpose, one needs to obtain a representation of `t1` data instances as perceived by the model trained for `t0`. This feature extraction process is done through a forward pass of `t1` data instances on the pre-trained CNN model, which generates a data embedding that can be fed to another machine learning method (*e.g.*, a Support Vector Machine, or SVM, for classification). In most cases, the embedding is defined by capturing and storing the activation values of a single layer close to the output [7]–[9], [13]–[15]. The rest of layers (*e.g.*, most convolutional layers) are discarded because these "are unlikely to contain a richer semantic representation than the later features" [13]. So far, this choice has been supported by comparisons between single-layer embeddings, where high-level layers have been shown to consistently outperform low-level layers [7], [8]. Nonetheless, this comparison is biased because convolutional and fully connected layers have different properties (*e.g.*, spatial information, output range and behaviour *etc.*), which suggests that activations extracted from both kinds of layers should be treated differently.

In contrast to single-layer performance comparisons, there are works demonstrating the ability of features from all layers to contribute to the characterisation of the data. In [16], an embedding using activations from all layers (*i.e.*, convolutional and fully connected layers) is built. While fully connected activations are extracted in a raw manner, convolutional ones are extracted using cross-convolutional-layer pooling. This technique consist of an special ROI pooling based on spatial activations from the following layer. In the experiments reported in [16], this makes multi-layer embeddings outperform single-layer embeddings. In a different work [17], authors study the ability of activations from all layers at characterising data. For a given target class and a trained feature, three disjoint behaviours are defined: the feature presence is relevant for characterising the class, the feature absence is relevant for characterising the class, the feature is irrelevant for the class. Features relevant by presence generate an abnormally high value for the class in the context of the dataset, while features relevant by absence generate an abnormally low value for the class in the same dataset context. Irrelevant features are the rest.

Beyond the layers used to generate the embedding and the characterisation of features extracted from those layers, there are other parameters that can affect the feature extraction process. Some of those are evaluated in [7], which includes parameters related with the architecture and training of the initial CNN (*e.g.*, network depth and width, distribution of training data, optimisation parameters), and parameters related with transfer learning process (*e.g.*, fine-tuning, spatial pooling and dimensionality reduction). Among the most frequently used transformations of deep embeddings is L2 normalisation [7], [8], and unsupervised feature reduction (*e.g.*, Principal Component Analysis or PCA) [7]–[9]. The quality of the resultant embedding is typically evaluated through the classification performance of an SVM, trained using the embedding representations of the train set, and tested using the embedding representations of the test set [7], [8]. Extracted features have also been combined with more sophisticated computer vision techniques, such as the constellation model [6] and Fisher vectors [18], with significant success.

III. Full-network Embedding

Transfer learning for feature extraction is used to embed a dataset `t1` in the representation language learnt for a task `t0`. To do so, one must forward pass each data instance of `t1` through the model pre-trained on `t0`, capturing the internal activations of the neurons composing the network. This is the *first step* of our method, but unlike most of previous contributions to feature extraction, we store the activation values generated on every convolutional and fully connected layer of the model to generate a full-network embedding. Following subsections explain in detail the following steps specific of our methodology. An end-to-end overview of the proposed embedding generation is shown in Fig. 1.

A. Spatial Average Pooling

During a feed-forward pass, each convolutional layer filter generates several activations for the given input as a result of the convolution process. These activations correspond to the application of the filter at the different possible locations within the input, and their number depends on both filter size and input size. In a feature extraction context, this larger number of activations significantly increases the dimensionality of the resulting embedding, which may often be counter-productive. Specially as the spatial information provided by the different activations may not be particularly relevant in a transfer learning setting (*i.e.*, one where the problem for which the filters were learnt is not the same as the problem where the filter is applied). Although complex solutions to maintain most of the spatial information have been proposed

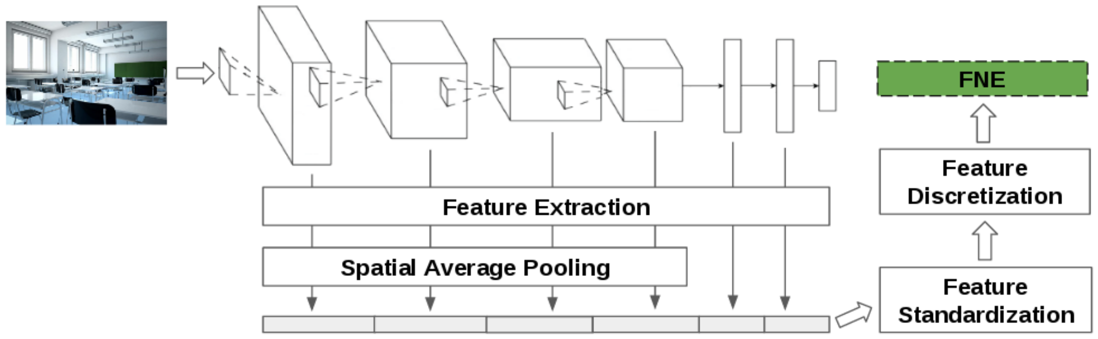

Fig. 1. Overview of the proposed out-of-the-box full-network embedding generation workflow.

[16] we use a more straight-forward solution by performing a *spatial average pooling* on each convolutional filter, such that a single value per filter is obtained by averaging all its spatially-depending activations [7], [8]. After this pooling operation, each feature in the embedding, from a single convolutional filter, corresponds to the degree with which this filter is found on average in the whole input, regardless of location. While losing spatial information, this solution maintains most of the embedding descriptive power, as all convolutional filters remain represented. A spatial average pooling on the filters of convolutional layers is the *second step* of our method. The values resulting from the spatial pooling are concatenated with the features from the fully connected layers into a single vector, to generate a complete embedding. In the case of the well-known `VGG16` architecture [19] this embedding vector is composed by 12,416 features.

B. Feature Standardisation

The features composing the embedding vector so far described are obtained from neurons of different type (*e.g.*, convolutional and fully connected layers) and location (*i.e.*, from any layer depth). These differences account for large variations in the corresponding feature activations (*e.g.*, distribution, magnitude, *etc.*). Since our method considers an heterogeneous set of features, a *feature standardisation* is needed. Our proposed standardisation computes the z-values of each feature, through the mean of train set activations and its standard deviation. This process transforms each feature value so that it indicates how separated the value is from the feature mean in terms of positive/negative standard deviations. In other words, the degree with which the feature value is atypically high (if positive) or atypically low (if negative) in the context of the `t1` dataset. A similar type of feature normalisation is used in deep network training (*i.e.*, batch normalisation) [20], but this is the first time this technique has been integrated in a feature extraction process. As discussed in §II, most feature extraction approaches apply an L2 norm by data instance, thus normalising by data instance instead of by feature. As shown in §IV, this approach provides competitive results, but is not appropriate when using features coming from many different layers. By using the z-values per feature, we use activations across the dataset as reference for normalisation. This balances each feature individually, which allows us to successfully integrate all types of features in the embedding. Significantly, this feature standardisation process generates a context dependent embedding, as the representation of each instance depends on the rest of instances being computed with it. For an example, consider how the features relevant for characterising a bird in a context of cars are different than the ones relevant for characterising the same bird in a context of other birds. Such a contextualised representation makes the approach more versatile, as it is inherently customised for each specific problem. After the feature standardisation, the final step of the proposed pipeline is a feature discretisation, which is described in §III-C.

C. Feature Discretisation

The embedding vectors we generate are composed of a large number of features (*e.g.*, 12,416 for the `VGG16` architecture). Exploring a representation space of such high-dimensionality is problematic for most machine learning algorithms, as it can lead to over-fitting and other issues related with the curse of dimensionality. A common solution is to use dimensionality reduction techniques like PCA [7], [14], [21]. We propose an alternative approach, which keeps the same number of features (and thus keeps the size and semantics of the representation language defined by the embedding) but reduces their expressiveness similarly to the quantization methodology followed in [21]. In detail, we discretise each standardised feature value to represent either an atypically low value (-1), a typical value (0), or an atypically high value (1). This discretisation is done by mapping feature values to the $\{-1, 0, 1\}$ domain by defining two thresholds ft^- and ft^+.

To find consistent thresholds, we consider the work [17], where a supervised statistical approach is used to evaluate the importance of CNN features for characterisation. Given a feature f and a class c, this work uses an empirical statistic to measure the difference between activation values of f for instances of c and activation values of f for the rest of classes in the dataset. This allows them to quantify the relevance of feature/class pairs for class characterisation. In their work, authors categorise these pairs in three different sets: *characteristic by absence*, *uncharacteristic* and *characteristic by presence*.

We use these three sets to find our thresholds ft^- and ft^+, by mapping the feature/class relevance of [17] to our corresponding feature/image activations. We do so on the datasets explored in [17]: *mit67*, *flowers102* and *cub200*, by computing the average values of the features belonging to each of the three sets. Fig. 2 shows the three resulting distributions of values for the *mit67* dataset. Clearly, a strong correlation exists between the supervised statistic feature relevance defined in [17] and the standardised feature values generated by the full-network embedding, as features in the characteristic by absence set correspond to activations which are particularly low, while features in the characteristic by presence set correspond to activations which are particularly high. We obtain the ft^- and ft^+ values through use of the Kolmogrov-Smirnov statistic on these distributions, which provides the maximum gap between two empirical distributions. Vertical dashed lines of Fig. 2 indicate these optimal thresholds for the *mit67* dataset, the rest are shown in Table I. To obtain a parameter free methodology, and considering the stable behaviour of the ft^+ and ft^- thresholds, we chose to set $ft^+ = 0.15$ and $ft^- = -0.25$ in all our experiments. Thus, after the step of feature standardisation, we discretise the values above 0.15 to 1, the values below -0.25 to -1, and the rest to 0.

This discretisation effectively reduces the amount of information each single feature provides from a real number to a three-valued one, which increases its generalisation capabilities. The effect can be analogous to an extreme reduction of the colour depth in an image, which results in a simpler, drawing-like image, which is a less detailed but a more generic representation of the original. The meaningfulness of the resulting discretised image depends on the suitability of the thresholds chosen between different colours. Similarly, the meaningfulness of the embeddings here obtained depends on the suitability of the thresholds chosen. The discover of a unique set of threshold values suitable for all the different datasets tested in §IV is a key point of our approach. This feature discretisation is the last step of the full-network embedding work-flow.

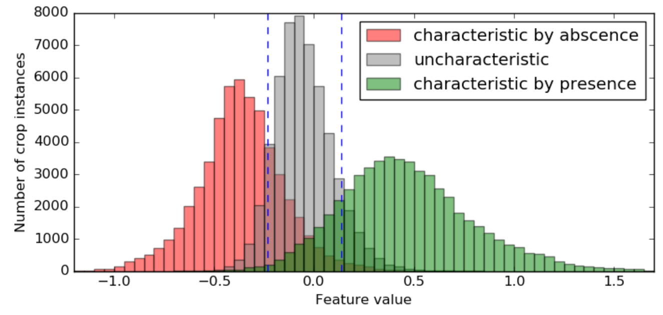

Fig. 2. For the *mit67* dataset, distribution of average standardised feature values for those features belonging to the sets identified in [17]. Vertical dashed lines mark the ft^- and ft^+ thresholds separating the two pairs of distributions as computed by the Kolmogrov-Smirnov statistic.

TABLE I
FEATURE VALUE THRESHOLDS ft^+ AND ft^- FOUND BY COMPUTING THE KOLMOGROV-SMIRNOV STATISTIC OF THE DISTRIBUTIONS EXEMPLIFIED IN FIG. 2.

Dataset	ft^+	ft^-
mit67	0.14	-0.23
cub200	0.20	-0.24
flowers102	0.15	-0.24

IV. EXPERIMENTS

In this section we consider the application of the previously defined full-network embedding to image classification problems. These experiments will allow us to understand the potential relevance and role of convolutional features when combined with fully connected features. We will also explore how changing the pre-trained models used to generate the embedding affects its performance for classification. This is an important study for expanding the applicability of transfer learning solutions to a wider range of domains.

A. Datasets

One of the goals of this paper is to identify a full-network feature extraction methodology which provides competitive results out-of-the-box. For that purpose, we evaluate the embedding proposed in §III-C on a set of 9 datasets which define different image classification challenges. The list includes datasets of classic object categorisation, fine-grained categorisation, and scene and textures classification. The disparate type of discriminative features needed to solve each of these problems represents a challenge for any approach which tries to solve them without any sort of tuning.

- The MIT Indoor Scene Recognition dataset [22] (*mit67*) consists of different indoor scenes to be classified in 67 categories. Its main challenge resides in the class dependence on global spatial properties and on the relative presence of objects.
- The Caltech-UCSD Birds-200-2011 dataset [23] (*cub200*) is a fine-grained dataset containing images of 200 different species of birds.
- The Oxford Flower dataset [24] (*flowers102*) is a fine-grained dataset consisting of 102 flower categories. The dataset contains only 20 samples per class for training.
- The Oxford-IIIT-Pet dataset [25] (*cats-dogs*) is a fine-grained dataset covering 37 different breeds of cats and dogs.
- The Stanford Dogs dataset [26] (*sdogs*) contains images from the 120 breeds of dogs found in ImageNet. The dataset is complicated by little inter-class variation, and large intra-class and background variation.
- The Caltech 101 dataset [27] (*caltech101*) is a classical dataset of 101 object categories containing clean images with low level of occlusion.
- The Food-101 dataset [28] (*food101*) is a large dataset of 101 food categories. Test labels are reliable but train images are noisy (*e.g.*, occasionally mislabeled).

TABLE II
PROPERTIES OF ALL CLASSIFICATION DATASETS COMPUTED

Dataset	#Images	#Classes	#Images (train)	#Images (test)	#Images per class (total)	#Images per class (train)	#Images per class (test)
mit67	6,700	67	5,360	1,340	100	77 - 83	17 - 23
cub200	11,788	200	5,994	5,794	41 - 60	29 - 30	12 - 30
flowers102	8,189	102	2,040	6,149	40 - 258	20	20 - 238
cats-dogs	7,349	37	3,680	3,669	184 - 200	93 - 100	88 - 100
sdogs	20,580	120	12,000	8,580	150 - 200	100	50 - 100
caltech101	9,146	101	3,060	2,995	31 - 800	30	1 - 50
food101	25,250	101	20,200	5,050	250	200	50
textures	5,640	47	3,760	1,880	120	80	40
wood	438	7	350	88	14 - 179	10 - 142	3 - 37

- The Describable Textures Dataset [29] (*textures*) is a database of textures categorised according to a list of 47 terms inspired from human perception.
- The Oulu Knots dataset [30] (*wood*) contains knot images from spruce wood, classified according to Nordic Standards. This dataset of industrial application is considered to be challenging even for human experts.

Details for these datasets are provided in Table II. This includes the train/test splits used in our experiments. In most cases we follow the train/test splits as provided by the dataset authors in order to obtain comparable results. A specific case is *caltech101* where, following the dataset authors instructions [27], we randomly choose 30 training examples per class and a maximum of 50 for test, and repeat this experiment 5 times. The other particular case is the *food101* dataset. Due to its large size, we use only the provided test set for both training and testing, using a stratified 5-fold cross validation. The same stratified 5-fold cross validation approach is used for the *wood* dataset, where no split is provided by the authors.

B. Experimental details

In this section we explain the details our experimental setup. To evaluate the consistency of our method out-of-the-box, we decide *not* to use additional data when available on the dataset (*e.g.*, image segmentation, regions of interest or other metadata), or to perform any other type of problem specific adaptation (*e.g.*, tuning hyper-parameters). Notice most of the methods we compare against use one or both of those performance improving techniques.

As source model for the feature extraction process (our `t0` tasks) we use the classical VGG16 CNN architecture [19] pre-trained on the Places2 scene recognition dataset [31] for the *mit67* experiments, and the same VGG16 architecture pre-trained on the *ImageNet 2012* classification dataset [32] for the rest. The motivation was to use a model pre-trained on a source task which is relatively similar to the target task, as this is the most optimistic deployment scenario.

With the generated embedding, a linear SVM is trained for classification with the default hyperparameter $C = 1$, using a one-vs-the-rest strategy. Standard data augmentation is used in the SVM training, using 5 crops per sample (4 corners + central) with horizontal mirroring (total of 10 crops per sample). At test time, all the 10 crops are classified, using a voting strategy to decide the label of each data sample.

Beyond the comparison with the current state-of-the-art (typically a thoroughly tuned model), we also compare our approach with the most popular feature extraction solution. As discussed in §II, an embedding is most frequently obtained by extracting the activations of one fully connected layer close to the output (`fc6` or `fc7` for the VGG16 model) and applying a L2 normalisation per data instance [7], [8], [13]. We call this our *baseline* method. The same pre-trained model used as source for the full-network embedding is used for the baseline. For both baselines (`fc6` and `fc7`), the final embedding is composed by 4,096 features. This is used to feed a SVM classifier, following the exact same methodology previously defined for both training and testing.

C. Results

The results of our classification experiments are shown in Table III. Performance is measured with average per-class classification accuracy. For each dataset we provide the accuracy provided by the baselines, by our method, and by the best method we found in the literature (*i.e.*, the state-of-the-art or SotA). For a proper interpretation of the performance gap between the SotA methods and ours, we further indicate if the SotA uses external data (beyond the `t1` dataset and the `t0` model) and if it performs fine-tuning.

Overall, our method outperforms the best baseline (`fc6`) by 2.2% accuracy on average. This indicates that the proposed full-network embedding successfully integrates the representations generated at the various layers, boosting performance in the process. The datasets where the baseline performs similarly or slightly outperforms the full-network embedding (*cub200*, *cats-dogs* and *sdogs*) are those where the target task `t1` overlaps with the source task `t0` (*e.g.*, *ImageNet 2012*). The largest difference happens for the *sdogs*, which is explicitly a subset of *ImageNet 2012*. In this sort of *pseudo* transfer learning problems, the fully connected layer used by the baseline methods has been partly optimized to solve the `t1` problem during the original CNN training phase. This explains why the baselines based on the fully connected layers are particularly competitive on these datasets. This *pseudo* transfer learning problems are included in our experiments because they define the most appropriate scenario for the baseline we

TABLE III
CLASSIFICATION RESULTS IN % OF AVERAGE PER-CLASS ACCURACY FOR THE BASELINES, FOR THE FULL-NETWORK EMBEDDING, AND FOR THE CURRENT STATE-OF-THE-ART (SOTA). ED: SOTA USES EXTERNAL DATA, FT: SOTA PERFORMS FINE-TUNING OF THE NETWORK.

Dataset	**mit67**	**cub200**	**flowers102**	**cats-dogs**	**sdogs**	**caltech101**	**food101**	**textures**	**wood**
Baseline `fc6`	80.0	65.8	89.5	89.3	78.0	91.4±0.6	61.4±0.2	69.6	70.8±6.6
Baseline `fc7`	81.7	63.2	87.0	89.6	79.3	89.7±0.3	59.1±0.6	69.0	68.9 ±6.8
Full-network	83.6	65.5	93.3	89.2	78.8	91.4±0.6	67.0±0.7	73.0	74.1±6.9
SotA	86.9 [5]	92.3 [10]	97.0 [5]	91.6 [6]	90.3 [5]	93.4 [33]	77.4 [4]	75.5 [18]	- -
ED	✓	✓	✓	✗	✓	✗	✗	✗	-
FT	✓	✓	✓	✓	✓	✓	✓	✗	-

could imagine. Even in this extremely adverse conditions for the FNE, the maximum advantage for the baseline is only 0.5%. The average advantage of the FNE over the baseline in the rest of experiments is 3.3%.

State-of-the-art performance is in most cases a few accuracy points above the performance of the full-network embedding (7.8% accuracy on average). These results are encouraging, considering that our method uses no additional data, requires no tuning of parameters and it is computationally cheap (*e.g.*, it does not require deep network training). The dataset where our full-network embedding is more clearly outperformed is the *cub200*. In this dataset [10] achieve a remarkable state-of-the-art performance by using lots of additional data (roughly 5 million additional images of birds) to train a deep network from scratch, and then fine-tune the model using the *cub200* dataset. In this case, the large gap in performance is caused by the huge disparity in the amount of training data used. A similar context happens in the evaluation of *food101*, where [4] use the complete training set for fine-tuning, while we only use a subset of the test set (see §IV-A for details). If we consider the results for the other 6 datasets, the average performance gap between the state-of-the-art and the full-network embedding is 4.2% accuracy on average.

Among the methods which achieve the best performance on at least one dataset, there is one which is not based on fine tuning a deep network. The work of [18] obtains the best results for the *textures* dataset by using a combination of bag-of-visual-words, Fisher vectors and convolutional filters. Authors demonstrate how this approach is particularly competitive on texture based datasets. The full-network embedding, while being more versatile, obtains an accuracy 2.5% lower in this specific domain.

The *wood* dataset is designed to be particularly challenging, even for human experts; according to the dataset authors the global accuracy of an experienced human sorter is about 75-85% [30], [34]. Currently, there are no reported results in average per-class accuracy for this dataset, so the corresponding values in Table III are left blank. Consequently, the results we report represent the current state-of-the-art to the best of our knowledge (74.1%±6.9 in average per-class accuracy). The best results previously reported in the literature for *wood* correspond to [15], which are 94.3% in global accuracy. However, the difference between average per-class accuracy and global accuracy is particularly relevant in this dataset, given the variance in images per class (from 3 to 37). To evaluate the average per-class accuracy, we tried our best to replicate the method of [15], which resulted in 71.0%±8.2 average per-class accuracy when doing a stratified 5-fold cross validation; a performance similar to the one obtained by our baseline method.

D. Study of Variants

In this section we consider removing and altering some of the components of the full-network embedding to understand their impact. First we remove feature discretisation, and evaluate the embeddings obtained after the feature standardisation step (FS). Secondly, we consider a partial feature discretisation which only maps values between ft^+ and ft^- to zero, and evaluate an embedding which keeps the rest of the original values ($\{-v, 0, v\}$). The purpose of this second experiment is to study if the increase in performance provided by the feature discretisation is caused by the noise reduction effect of mapping frequent values to 0, or if it is caused by the space simplification resultant of mapping all activations to only three values.

As shown in Table IV, the full-network embedding outperforms all the other variants, with the exceptions of *flowers102* and *cats-dogs* where FS is slightly more competitive (+0.8% and +0.7% accuracy) and *caltech101* where the best is $\{-v, 0, v\}$ by 0.5% accuracy . The noise reduction variant (*i.e.*, $\{-v, 0, v\}$) outperforms the FS variant in 5 out of 9 datasets. The main difference between both is that the former sparsifies the embeddings by transforming *typical* values to zeros, with few informative data being lost in the process. The complete feature discretisation done by the full-network model (*i.e.*, $\{-1, 0, 1\}$) further boosts performance, outperforming the $\{-v, 0, v\}$ embedding on 7 of 9 datasets. This shows the potential benefit of reducing the complexity of the embedding space.

The feature discretisation also has the desirable effect of reducing the training cost of the SVM applied on the resulting embedding. Using the FS embedding as control (the slowest of all variants), the $\{-v, 0, v\}$ embedding trains the SVM between 3 and 13 times faster depending on the dataset, while the full-network embedding with its complete discretisation trains between 10 and 50 times faster. Significantly, all three embeddings are composed by 12,416 features. For comparison, the baseline method, which uses shorter embeddings of 4,096 features, trains the SVM between 100 and 650 times faster

TABLE IV

MEAN PER-CLASS ACCURACY IN % OBTAINED BY THE FULL-NETWORK EMBEDDING, WHEN NOT PERFORMING FEATURE DISCRETISATION (FS), AND WHEN ONLY DISCRETIZING THE VALUES BETWEEN THRESHOLDS ($\{-v, 0, v\}$).

Dataset	**mit67**	**cub200**	**flowers102**	**cats-dogs**	**sdogs**	**caltech101**	**food101**	**textures**	**wood**
FS	81.0	64.9	94.1	89.9	77.3	91.5	65.3	69.6	71.5
$\{-v, 0, v\}$	82.0	64.8	93.2	89.3	77.4	92.0	66.6	70.1	70.6
Full-network	83.6	65.5	93.3	89.2	78.8	91.5	67.0	73.0	74.1

TABLE V

CLASSIFICATION RESULTS IN % AVERAGE PER-CLASS ACCURACY OF THE BASELINE AND THE FULL-NETWORK EMBEDDING WHEN USING A NETWORK PRE-TRAINED ON *ImageNet 2012* FOR *mit67* AND ON *Places2* FOR THE REST.

Dataset	**mit67**	**cub200**	**flowers102**	**cats-dogs**	**sdogs**	**caltech101**	**food101**	**textures**	**wood**
Baseline `fc7`	72.2	23.6	73.3	38.7	24.7	72.0	40.5	55.8	65.3
Full-network	75.5	35.5	88.7	56.2	37.8	80.0	55.9	65.1	74.0

than the FS. For both the baseline and the full-network embeddings, training the SVM takes a few minutes on a single CPU.

A different variation we consider is to use an inappropriate task `t0` as source for generating the baseline and the full-network embeddings. This tests the robustness of each embedding when using an ill-suited pre-trained model. These experiments are particularly relevant for most real world applications, where labeled data is limited, and the existence of a pre-trained model similar to the target task is unlikely.

In our experiments we use the model pre-trained on *ImageNet 2012* for generating the *mit67* embeddings, and the model pre-trained on *Places2* for the rest of datasets. This is the opposite combination than the one used in §IV-C. Table V shows that the full-network embedding is much more robust in this context, with an average decrease in accuracy of 16.4%, against a 24.6% decrease of the baseline. These results remark the limitation of the baseline, caused by its own dependency on late layer features. Finally, we also considered using different network depths, a parameter also analysed in [7]. We repeated the full-network experiments using the VGG19 architecture instead of the VGG16, and found performance differences to be minimal (maximum difference of 0.3%) and inconsistent.

V. CONCLUSIONS

In this paper we describe a feature extraction process which leverages the information encoded in all the features of a deep CNN. The full-network embedding extracts information from all layers, introduces the use of feature standardisation and of a novel feature discretisation methodology based on the work by [17]. The former provides extended encoded visual information of images through different levels of complexity, while the latter provides context-dependent embeddings, which adapt the representations to the problem at hand. Finally, the proposed feature discretisation process reduces noise and regularises the embedding space while keeping the size of the original representation language.

The resultant full-network embedding is shown to outperform single-layer embeddings in several classification tasks. It outperforms the best baseline by 2.2% accuracy on average, even though 3 of the 9 datasets used for comparison are strongly biased towards the baseline (since these are direct subsets of the source task *ImageNet 2012*). In these 3 particularly unfavourable tests the advantage of the best baseline is at most 0.5%, showcasing the competitiveness of the proposed method under optimal transfer conditions for the baseline.

One of the most appealing properties of the full-network embedding is its robustness to the use of ill-suited pre-trained models in image classification. This is the most common scenario in real-world settings, when there is rarely a large dataset similar to the target task to pre-train with . When a dissimilar source task is used to pre-train the model, the difference with the baseline grows from 2.2% accuracy on average to a remarkable 11.4%. On top of that, the full-network embedding is parameter-free, capable of providing results out-of-the-box.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Z. Xu, S. Huang, Y. Zhang, and D. Tao, "Augmenting strong supervision using web data for fine-grained categorization," in *Proc. IEEE International Conference on Computer Vision*, 2015, pp. 2524–2532.

[3] S. Branson, G. Van Horn, S. Belongie, and P. Perona, "Bird species categorization using pose normalized deep convolutional nets," *arXiv preprint arXiv:1406.2952*, 2014.

[4] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, and Y. Ma, "Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment," in *International Conference on Smart Homes and Health Telematics*. Springer, 2016, pp. 37–48.

[5] W. Ge and Y. Yu, "Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning," *arXiv preprint arXiv:1702.08690*, 2017.

[6] M. Simon and E. Rodner, "Neural activation constellations: Unsupervised part model discovery with convolutional networks," in *Proc. IEEE International Conference on Computer Vision*, 2015, pp. 1143–1151.

[7] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of transferability for a generic convnet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 9, pp. 1790–1802, 2016.

[8] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 806–813.

[9] Y. Gong, L. Wang, R. Guo, and S. Lazebnik, "Multi-scale orderless pooling of deep convolutional activation features," in *European conference on computer vision*. Springer, 2014, pp. 392–407.

[10] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei, "The unreasonable effectiveness of noisy data for fine-grained recognition," in *European Conference on Computer Vision*. Springer, 2016, pp. 301–320.

[11] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[12] M. Long, Y. Cao, J. Wang, and M. I. Jordan, “Learning transferable features with deep adaptation networks.” in *ICML*, 2015, pp. 97–105.

[13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A deep convolutional activation feature for generic visual recognition.” in *International Conference on Machine Learning*, vol. 32, 2014, pp. 647–655.

[14] A. Mousavian and J. Kosecka, “Deep convolutional features for image based retrieval and scene categorization,” *arXiv preprint arXiv:1509.06033*, 2015.

[15] R. Ren, T. Hung, and K. C. Tan, “A generic deep-learning-based approach for automated surface inspection,” *IEEE Trans. Syst., Man, Cybern.*, 2017.

[16] L. Liu, C. Shen, and A. van den Hengel, “The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4749–4757.

[17] D. Garcia-Gasulla, F. Parés, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, “On the behavior of convolutional nets for feature extraction,” *arXiv preprint arXiv:1703.01127*, 2017.

[18] M. Cimpoi, S. Maji, and A. Vedaldi, “Deep filter banks for texture recognition and segmentation,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3828–3836.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[20] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[21] M. Carvalho, M. Cord, S. Avila, N. Thome, and E. Valle, “Deep neural networks under stress,” in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 4443–4447.

[22] A. Quattoni and A. Torralba, “Recognizing indoor scenes,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 413–420.

[23] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, “The caltech-ucsd birds-200-2011 dataset,” 2011.

[24] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *2008 Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, 2008, pp. 722–729.

[25] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, “Cats and dogs,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3498–3505.

[26] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, “Novel dataset for fine-grained image categorization: Stanford dogs,” in *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, vol. 2, 2011.

[27] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories,” *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[28] L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101–mining discriminative components with random forests,” in *European Conference on Computer Vision*. Springer, 2014, pp. 446–461.

[29] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, “Describing textures in the wild,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3606–3613.

[30] O. Silvén, M. Niskanen, and H. Kauppinen, “Wood inspection with non-supervised clustering,” *Machine Vision and Applications*, vol. 13, no. 5, pp. 275–285, 2003.

[31] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva, “Places: An image database for deep scene understanding,” *arXiv preprint arXiv:1610.02055*, 2016.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[33] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in *European Conference on Computer Vision*. Springer, 2014, pp. 346–361.

[34] J. Lampinen and S. Smolander, “Wood defect recognition with self-organizing feature selection,” in *Photonics for Industrial Applications*. International Society for Optics and Photonics, 1994, pp. 385–395.